Seamless Service Provision in P2P Service Overlays

Kamill Panitzek*, Immanuel Schweizer*, Muhammad Ikram† and Max Mühlhäuser*
*Telecooperation Group †Peer-to-Peer Networks Group
Technische Universität Darmstadt

I. Introduction

Different peer-to-peer (P2P) systems have been used over the past decade to share and distribute data. Examples are file sharing systems, Voice over IP (VoIP) communication and video streaming. They all provide means to discover and distribute data or data streams.

Our vision for the next generation of P2P systems is the additional distribution of applications and services. This enables a new family of P2P systems so called P2P service overlays. P2P service overlays enable the discovery and distribution but also the execution and composition of services (i.e. [1]). P2P service overlays pose a number of questions. Services need to be discovered at changing locations, execution states need to be transferred and due to device heterogeneity services are not executable on any device. Also, robustness and load balancing as well as other problems need to be tackled.

Throughout the presented work we will focus on the problem of *seamless service provision*. When services are copied or moved through the network this must be hidden from the service consumer. The service operation needs to be stable until the handover is completed.

Before we can propose a possible solution we will define the main components and methods that compose a P2P service overlay. Afterwards we will give a solution to the problem of seamless service provision by building a reliable, robust, and interruption free first responder communication service. First responders are mobile and have only limited resources available. Also the network is highly unreliable and decentralized. Providing services using a P2P service overlay will promote service distribution, load balancing and service localization and will lead to more reliable service provision under such harsh conditions.

II. P2P Service Overlay

P2P systems provide the basic functionality to connect peers with each other using an application overlay and to *distribute* and *discover* data items. Additionally P2P service overlays provide the functionality to invoke executable data items, called services. P2P service overlays are composed using three major building blocks depicted in Figure 1. **Peers** are devices connected to each other thus constructing the P2P overlay. Peers are best described by their given software and hardware resources. These resources include for example operating systems, runtime environments, CPU, memory, storage capacity, network connectivity and also energy supply. Users interact with the system over their respective peer. This includes all user behaviors, e.g. real-world movement as well as service request and service interaction.

Fig. 1. Main building blocks of P2P service overlays

Mobile objects describe any object stored in the network that can be distributed using the P2P system. These mobile objects are further categorized in *data items* or *data streams* and *services* or *applications*. P2P service overlays concentrate on the distribution of services as mobile objects.

Every peer contains a **runtime environment** where mobile objects are stored and executed. To protect the peer's hardware and software from malicious code this runtime environment might be implemented as virtual machines. It can be understood as a middleware for service provision.

We identified basic functional and non-functional requirements of P2P service overlays. Furthermore, we also identified possible components and solutions covering these requirements. As described above, peers must have the ability to *distribute* and *discover* mobile objects inside the system. A **service registry** provides this functionality and can be realized using a classical distributed hash table (DHT) (i.e. [2]) or a hierarchical P2P system as proposed by Kellerer et al. [3] for mobile P2P networks.

Particularly for the *execution* of a service it must be decided where to execute it, either on a remote peer or locally on the peer requesting that service. Due to device heterogeneity, peers' resources and the resource requirements of mobile objects have to be matched. Devices cannot host services requiring other resources than are available. To provide this functionality a decentralized **matching algorithm** is needed.

As known from classical P2P systems, *availability* of mobile objects is very important and can be compromised with high churn rates (peers connecting to and disconnecting from the system). Adding peer mobility further increases this problem due to possible connection loss. Therefore, a **replication mechanism** is needed to automatically replicate popular services, thus increasing the availability of mobile objects.

Reaching high *robustness* to dynamic changes in the environment and to peer mobility is of great importance to P2P service overlays in disaster response. In such environments, the applications have to be mobile. The code as well as the current

execution state has to be transferred efficiently between peers. From a users' point of view this transfer has to be unnoticeable requiring *seamless service provision*. This can be achieved by using **code migration** and **persistent socket connections**.

To further improve robustness of the system, peers must be prevented from overloading, so that the hosted service (mobile object) can be provided. A **pro-active load balancing mechanism** has to identify possibly overloaded peers and replicate or migrate hosted mobile objects onto another peer.

III. Seamless Service Provision

P2P service overlays pose different problems that need to be solved. Here we focus on the challenge of seamless service provision. When migrating a service during runtime it is crucial that the service is not interrupted. This is especially true in disaster response where an interrupted service is not only decreasing user experience but safety as well. Usually when migrating a service during runtime the peer initiating the migration process pauses the service, transfers the code and the execution state to another peer where the service is resumed. This procedure will cause the service to interrupt for at least the time it takes to transfer the service between peers.

Our solution to this problem is to replicate the service and synchronize the execution states. This means that the service code and the execution state are copied to another peer and executed there. During the handover procedure all state changes are propagated to the new server peer as update messages. Once the services hosted on both peers are synchronized, the service on the new peer will take over and the old service initiates the shutdown procedure. The old service will disconnect the users only if the new service is ready to take over at the exact same state basically implementing a software handover protocol.

During disaster response voice communication is one of the most critical services. In most contemporary group voice communication software, like Skype[1] or TeamSpeak[2], a server (or a peer) provides the communication service. The voice packets are sent to the server and then redistributed to participating clients. In terms of service overlays we call this type of service an *interactive service*, since client peers are connected to and constantly interacting with the service. Migrating such an interactive voice service is only possible if the communication does neither stutter nor disconnect. Not only the state needs to be transferred but also the connections between clients and the server peer. They have to be reestablished once the service is started on the new peer. Thus all peers are first connected to both services. After the new service provider peer sends a take-over message to all reconnected clients, the clients start to transmit their voice packets to the new provider and disconnect from the old one. This introduces timing problems, because it might be possible that one client still sends its voice packets to the old server while other peers already send their packets to the new one. These packets are then propagated from the old server peer to the new server using update messages. Sequence numbers are introduced to allow for a seamless transmission of these packets.

[1]Skype: http://www.skype.com

[2]TeamSpeak: http://www.teamspeak.com

IV. Framework Implementation

We implemented a framework for P2P service overlays using Java and parts of the MundoCore middleware [4]. Persistent sockets were implemented using a method similar to MobileSocket [5] but with further enhancements and with a TCP channel for control messages and an UDP channel for data transmission. Furthermore, we used OpenChord[3] as service registry based on the Chord DHT by Stoica et al. [2]. This framework enabled us to realize a voice chat service prototype and distribute it as mobile object.

The first results of the evaluation are promising. First experiments conducted with our prototype are free of voice interruptions during the migration process. Currently the control message channel is paused during the migration process but the data channel remains open. Data transmission then switches to the new server peer once the take-over message is received. This enables an interruption free voice chat service. As for the migration process we also are working to further reduce the migration time with special compression mechanisms to pack the current execution state more efficiently.

V. Conclusion

In the presented work we focused on seamless service provision in P2P service overlays. We provided a seamless voice chat service prototype focusing on disaster response scenarios. Another requirement we mentioned is availability of services and will be discussed specifically in P2P based games by Muhammad Ikram. In our future work we will further enhance the migration process and also focus on distributed service discovery and invocation as well as on decentralized matching algorithms.

Acknowledgments

This work has been funded by the DFG research unit 733 QuaP2P. We would like to thank Elias Weingärtner from RWTH Aachen for his supportive comments.

References

[1] X. Gu, K. Nahrstedt, and B. Yu, *Spidernet: an integrated peer-to-peer service composition framework*. IEEE, 2004.

[2] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, ser. SIGCOMM '01. New York, NY, USA: ACM, 2001, pp. 149–160.

[3] W. Kellerer, Z. Despotovic, M. Michel, Q. Hofstatter, and S. Zols, "Towards a Mobile Peer-to-Peer Service Platform," in *2007 International Symposium on Applications and the Internet Workshops*. IEEE, Jan. 2007, pp. 2–2.

[4] E. Aitenbichler, J. Kangasharju, and M. Mühlhäuser, "MundoCore: A light-weight infrastructure for pervasive computing," *Pervasive and Mobile Computing*, vol. 3, no. 4, pp. 332–361, Aug. 2007.

[5] T. Okoshi, M. Mochizuki, Y. Tobe, and H. Tokuda, "MobileSocket: toward continuous operation for Java applications," in *Proceedings Eight International Conference on Computer Communications and Networks (Cat. No.99EX370)*. IEEE, 1999, pp. 50–57.

[3]Open-Chord: http://sourceforge.net/projects/open-chord/